Ontology lexica and automatic grammar generation

Philipp Cimiano · Christina Unger
Semantic Computing Group
CITEC, Bielefeld University
These slides are not the original ones produced by prof. Philipp Cimiano.

They have been edited by Manuel Fiorelli (fiorelli@info.uniroma2.it).
So far we have:

1. Ontologies
2. LTAG/DUDES grammars aligned to the ontology

But…

1. Who is going to write those grammars for new domains?
2. What if we want to produce different grammar formats (e.g. CG, HPSG, GF, etc.)?

**Goal:** automatize the task of grammar generation
Today
lemon: A model for the lexicon-ontology interface

Automatic grammar generation
  Verbs
  Proper names
  Nouns
  Prepositions
  Adjectives
Outline

lemon: A model for the lexicon-ontology interface

Automatic grammar generation
Verbs
Proper names
Nouns
Prepositions
Adjectives
In order to generate grammars for a given ontology, we need to enrich the ontology with lexical and linguistic information.
In order to generate grammars for a given ontology, we need to enrich the ontology with lexical and linguistic information.

An ontology lexicon specifies how ontology concepts correspond to natural language expressions. This can support:

1. **Interpretation** (knowing how natural language expressions should be interpreted with respect to a given ontology)

2. **Generation** (knowing how ontology concepts can be verbalized)
Semantics by reference

Semantics by reference (McCrae et al. 2012)

Lexicon and ontology are clearly separated. The meaning of lexical entries is specified by pointing to elements in the ontology.

1. That is, the ontology can live without the lexicon, while the lexicon depends on a given ontology.

1. One ontology can be connected to several lexica (e.g. for different languages).
lemon (Lexicon model for ontologies) is a meta-model for describing lexica with RDF.

Most importantly, lemon is agnostic w.r.t linguistic categories.

- It has general concepts like `lexical entry'.
- It does not have specific concepts like `noun', 'intransitive verb', `singular', etc.

lemon is developed in collaboration between CITEC (Bielefeld), DERI (Galway), UPM (Madrid), DFKI (Saarbrücken), and TU Delft.
The lemon model
Lexicon
  language:String

Form
  writtenRep:String

Linguistic Ontology

entry

LexicalEntry*
  (Word, Phrase, Part)

lexicalForm
  canonicalForm
  otherForm

Frame
  synBehavior

Argument
  synArg

LexicalSense
  marker
  semArg

Ontology

reference

subjOfProp
  objOfProp
  isa

* LexicalEntry has three subclasses: Word, Phrase, Part
† PhraseTerminals are Arguments or Components
‡ marker can also refer to linguistic ontology
Lemon core concepts

1. **Lexicon**
   The lexicon is itself a resource and has one associated language.

2. **Lexical Entry**
   Each entry is a resource and is split into three subclasses: Word, Phrase, Part.

3. **Form**
   Each entry has a number of forms. Each form has a number of written representations.

4. **Lexical Sense**
   Each entry has a number of senses. Each sense has a reference pointing to an ontological symbol (entity, class or relation).
Lexica and entries

Entries are indicated with the entry property.

```turtle
@prefix : <http://www.example.org/lexicon/> .
@prefix lemon: <http://lemon-model.net/lemon#> .
:lexicon a lemon:Lexicon ;
    lemon:language "en" ;
    lemon:entry :team,
        :match,
        :goal,
        :win,
        ... .
```

An ISO 639 language code
Forms

@prefix : <http://www.example.org/lexicon/> .
@prefix lemon: <http://lemon-model.net/lemon#> .

:team  a lemon:Word;
   lemon:canonicalForm  [ lemon:writtenRep  "team"@en ] ;
   lemon:otherForm      [ lemon:writtenRep  "teams"@en ] .

:win   a lemon:Word ;
   lemon:canonicalForm  [ lemon:writtenRep  "win"@en ] ;
   lemon:otherForm      [ lemon:writtenRep  "wins"@en ] ;
   lemon:otherForm      [ lemon:writtenRep  "won"@en  ] ;
   lemon:otherForm      [ lemon:writtenRep  "winning"@en ] .
**Forms**

- win : Word
  - canonical form: Form
    - writtenRep = "win"@en
  - other form: Form
    - writtenRep = "wins"@en
  - other form: Form
    - writtenRep = "won"@en
  - other form: Form
    - writtenRep = "winning"@en

l:emon: A model for the lexicon-ontology interface
Senses and references

1. :team lemon:sense [ lemon:reference
2.  <http://sc.cit-ec.uni-bielefeld.de/ontologies/soccer#Team> ].

3.

4. :win lemon:sense [ lemon:reference

---

**Diagram:**

```
win : Word

<table>
<thead>
<tr>
<th>sense</th>
</tr>
</thead>
</table>

: LexicalSense

<table>
<thead>
<tr>
<th>reference</th>
</tr>
</thead>
</table>

<http://sc.cit-ec.uni-bielefeld.de/ontologies/soccer#winner>
```
Frames

**Syntactic arguments:** Each word has a subcategorization frame.

- **intransitive:** $X$ won ($X = \text{subject}$)
- **transitive:** $X$ won $Y$ ($X = \text{subject}$, $Y = \text{direct object}$)

**Semantic arguments:** Each ontological relation has a number of arguments.

- $<\text{subject}>\ \text{soccer:winner}\ <\text{object}>$
Frames

**Syntactic arguments:** Each word has a subcategorization frame.

- intransitive: $X \text{ won ($X = \text{ subject}$)}$
- transitive: $X \text{ won } Y$ ($X = \text{ subject, } Y = \text{ direct object}$)

**Semantic arguments:** Each ontological relation has a number of arguments.

- $<\text{subject}> \text{ soccer:winner } <\text{object}>$.

We also need to specify the **correspondence** between syntactic and semantics arguments.

- Some team won some game.
- $<\text{soccer:Match}> \text{ soccer:winner } <\text{soccer:Team}>$. 

**lemon:** A model for the lexicon-ontology interface
Linguistic ontology

: Frame
  synBehavior
  synArg
  : Argument
    synArg
    subjOfProp
    : Argument
    objOfProp
    : LexicalSense
      sense
      reference
      <soccer:winner>
Linguistic ontology

lemon provides a general model but stays agnostic w.r.t. linguistic theories. Hence there is no way to indicate the syntactic roles of the arguments (subject, direct object, indirect object, etc.).

For specifics, lemon thus needs to be extended with a linguistic ontology.

[We need version 2.0: http://lexinfo.net/ontology/2.0/lexinfo]

- LexInfo
- ISOcat
- ...

lemon: A model for the lexicon-ontology interface
Linguistic ontology

Frame

Argument

synBehavior

Argument

LexicalSense

<soccer:winner>

lexinfo:subject

lexinfo:directObject

subjOfProp

objOfProp

sense

reference
Lexical properties

All LexInfo's lexical properties are subproperties of lemon's property.

- `lexinfo:partOfSpeech`
  - `lexinfo:noun`, `lexinfo:verb`, `lexinfo:adjective`, …
- `lexinfo:number`
  - `lexinfo:singular`, `lexinfo:plural`
- `lexinfo:person`
  - `lexinfo:firstPerson`, `lexinfo:secondPerson`, `lexinfo:thirdPerson`
- `lexinfo:tense`
  - `lexinfo:present`, `lexinfo:past`
- …
A simple lexical entry

: TransitiveFrame
  synBehavior
  directObject
  subject

: Argument
  subjOfProp

: Argument

: Argument

: LexicalSense
  sense
  reference

: Word
  partOfSpeech = verb

: Form
  writtenRep = "win"@en
tense = present

: Form
  writtenRep = "wins"@en
tense = present person = thirdPerson number = singular

: Form
  writtenRep = "won"@en
verbFormMood = participle
aspect = perfective

: Form
  writtenRep = "winning"@en
verbFormMood = gerundive

lemon: A model for the lexicon-ontology interface
A simple lexical entry

```
:win a lemon:LexicalEntry ;

  lexinfo:partOfSpeech lexinfo:verb ;
  lemon:synBehavior [ rdf:type lexinfo:TransitiveFrame ;
                   lexinfo:subject _:arg1 ;
                   lexinfo:directObject _:arg2 ] ;

  lemon:sense [ lemon:reference
               <http://sc.cit-ec.uni-bielefeld.de/ontologies/soccer#winner> ;
               lemon:subjOfProp _:arg2 ;
               lemon:objOfProp _:arg1 ] ;

  lemon:canonicalForm [ lemon:writtenRep "win"@en ;
                       lexinfo:tense lexinfo:present] ;
  lemon:otherForm [ lemon:writtenRep "wins"@en ; ... ] .
```

lemon: A model for the lexicon-ontology interface
Most common frames (verbs)

1. **Intransitive**
   - Arguments: subject
   - Example: win

2. **Transitive**
   - Arguments: subject, direct object
   - Example: win (smth)

3. **Intransitive PP**
   - Arguments: subject, prepositional object
   - Example: win against (so)

4. **Transitive PP**
   - Arguments: subject, direct object, prepositional object
   - Example: win (smth) against (so)
Creating lexica with lemon

- lemon source: [http://monnetproject.deri.ie/lemonsourc](http://monnetproject.deri.ie/lemonsourc)
- Java API: [http://www.lemon-model.net/api.html](http://www.lemon-model.net/api.html)
Outline

lemon: A model for the lexicon-ontology interface

Automatic grammar generation
- Verbs
- Proper names
- Nouns
- Prepositions
- Adjectives
Generating grammars from lexica

For every lexicon entry:
Based on the part of speech and syntactic frame…
- intransitive verb (with PP)
- transitive verb (with PP)
- noun (with PP)
- adjective

…extract relevant properties (word forms, required arguments, semantic restrictions, etc.) and build corresponding grammar entries.
lemon: A model for the lexicon-ontology interface

Automatic grammar generation
- Verbs
- Proper names
- Nouns
- Prepositions
- Adjectives
Example: to win

<table>
<thead>
<tr>
<th>: Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>writtenRep = &quot;wins&quot;@en</td>
</tr>
<tr>
<td>tense = present</td>
</tr>
<tr>
<td>person = thirdPerson</td>
</tr>
<tr>
<td>number = singular</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>: Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>partofSpeech = verb</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>: LexicalSense</th>
</tr>
</thead>
<tbody>
<tr>
<td>reference</td>
</tr>
</tbody>
</table>

| /http://sc.cit-ec.uni-bielefeld.de/ontologies/soccer#winner/ |
| : Argument |
| : Argument |

<table>
<thead>
<tr>
<th>subjOfProp</th>
</tr>
</thead>
<tbody>
<tr>
<td>objOfProp</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>: TransitiveFrame</th>
<th>synBehavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>directObject</td>
<td>subject</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: to win

S

DP₁↓

V

DP₂↓

wins

: Form
writtenRep = "wins"@en
tense = present person =
thirdPerson number =
singular

: TransitiveFrame
synBehavior

directObject
subject

: Argument
: Argument

: Word
partOfSpeech = verb

: LexicalSense
sense
reference

http://sc.cit-ec.uni-bielefeld.de/ontologies/soccer#winner/

Automatic grammar generation  Verbs
Example: to win

```
: Form
  writtenRep = "wins"@en
  tense = present person = thirdPerson number = singular

: TransitiveFrame
  synBehavior
  directObject
  subject
  subjOfProp
  objOfProp

: Argument
  : Argument

: Word
  partOfSpeech = verb

: LexicalSense
  sense
  reference

: TransitiveFrame
  synBehavior
  directObject
  subject
  subjOfProp
  objOfProp

: Argument
  : Argument

S
  DP_1↓
  VP
  V
  DP_2↓

wins

http://sc.cit-ec.uni-bielefeld.de/ontologies/soccer#winner

soccer:winner(y, x)
(x, DP_1), (y, DP_2)
```
Grammar entries for transitive verbs

Present (3P singular, others) and past:

\[
S \rightarrow \text{DP}_1 \downarrow \text{VP} \\
\text{VP} \rightarrow \text{V} \downarrow \text{DP}_2 \\
\text{V} \rightarrow \text{win} \rightarrow \text{wins} \rightarrow \text{won} \\
\]

<table>
<thead>
<tr>
<th>soccer:winner(y, x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x, \text{DP}_1), (y, \text{DP}_2)</td>
</tr>
</tbody>
</table>
Grammar entries for transitive verbs

Passive (using past participle):

S
    ↓
DP₂ ↓ VP
    ↓
V VP
    ↓
  V VP
    ↓
is are was were won
    ↓
  P DP₁ ↓
    by

<table>
<thead>
<tr>
<th>soccer:winner(y, x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x, DP₁), (y, DP₂)</td>
</tr>
</tbody>
</table>
Grammar entries for transitive verbs

Gerundive:

```
NP
   NP*
   ADJ
   ADJ
   DP2
winning
```

```
<table>
<thead>
<tr>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>soccer:winner(y, x)</td>
</tr>
<tr>
<td>(y, DP2)</td>
</tr>
</tbody>
</table>
```
Gerundive:

```
NP
  NP* ADJ
    ADJ PP
      won P DP1↓
        by
```

```
soccer:winner(y, x)
(x, DP1)
```
Grammar entries for transitive verbs

Relative clauses:

```
NP
  NP*
    S
      COMP
        that
          V
            DP
              win
              wins
              won

x
soccer:winner(y, x)
(y, DP2)
```
Grammar entries for transitive verbs

Relative clauses:

```
NP
  NP*  S
    COMP  VP
      that  V  VP
           is  V  PP
              /  P  DP₁
               won  by
```

```
<table>
<thead>
<tr>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>soccer:winner(y, x)</td>
</tr>
<tr>
<td>(x, DP₁)</td>
</tr>
</tbody>
</table>
```
Intransitive verbs

Example:

S
   ▲
  / \  
DP₁ ↓ VP
 /   |
V    |
|   lost

| y, t |
| soccer:loser(y, x) |
| time : hasEnd(y, t) |
| t < now |
| (x, DP₁) |
Examples:

1. win against (so.)
2. score from (smth.)
3. win (smth.) against (so.)
4. substitute (so.) for (so.)

Are prepositional arguments subcategorized or do they adjoin?
Intransitive PP (subcategorized)

S

DP₁ ↓

VP

V

win

PP

P

against

DP₃ ↓

<table>
<thead>
<tr>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>soccer:winner(z, x)</td>
</tr>
<tr>
<td>soccer:loser(z, y)</td>
</tr>
<tr>
<td>(x, DP₁), (y, DP₃)</td>
</tr>
</tbody>
</table>
Intransitive PP (adjoining)

S
  DP₁ ↓ VP
    V
      win

VP *
  VP
    PP
      P
        DP₃ ↓
          s
            soccer:stadium(z, s)
            soccer:city(s, y)
            (y, DP₃)

Table:
<table>
<thead>
<tr>
<th>z</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>soccer:winner(z, x)</td>
<td></td>
</tr>
<tr>
<td>(x, DP₁)</td>
<td></td>
</tr>
</tbody>
</table>
lemon: A model for the lexicon-ontology interface

Automatic grammar generation

- Verbs
- Proper names
- Nouns
- Prepositions
- Adjectives
In our case, entities are represented by URIs. For example:

```
DP
| name
---|---
| x  |
| x  |
```

```
DP
| Uruguay
---|---
| x  |
| x  |
```

\[ x = \text{soccer:Uruguay} \]
lemon: A model for the lexicon-ontology interface

Automatic grammar generation

Verbs
Proper names
Nouns
Prepositions
Adjectives
Nouns

1. semantically simple nouns (usually expressing atomic classes)
   - goal, match, team

2. semantically complex nouns (not expressing atomic classes)
   - opener, player, coach

3. relational nouns (usually expressing a property)
   - winner, wife, distance, capital
Semantically simple nouns

1: goal a lemon:LexicalEntry; 
2: lexinfo:partOfSpeech lexinfo:noun; 
3: lemon:canonicalForm [ lemon:writtenRep "goal"@en; 
4: lexinfo:number lexinfo:singular] ;
5: lemon:otherForm [ lemon:writtenRep "goals"@en; 
6: lexinfo:number lexinfo:plural] ;
7: lemon:sense [ lemon:reference soccer:Goal ].

Automatic grammar generation

Nouns

NP

<table>
<thead>
<tr>
<th>goal(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

DP

<table>
<thead>
<tr>
<th>goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>x</td>
</tr>
</tbody>
</table>

soccer:Goal(x)

soccer:Goal(x)
Semantically complex nouns

```prolog
:player a lemon:LexicalEntry ;
lexinfo:partOfSpeech lexinfo:noun ;
...
lemon:sense
  [ lemon:subsense [ lemon:reference soccer:Person ] ,
  [ lemon:reference soccer:role ;
```

**NP**
- player(s)
  - x
  - r
  - soccer:Person(x)
  - soccer:role(x, r)
  - soccer:PlayerRole(r)

**DP**
- players
  - x, r
  - soccer:Person(x)
  - soccer:role(x, r)
  - soccer:PlayerRole(r)
Relational nouns

```
1  :winner a lemon:LexicalEntry ;
2  lexinfo:partOfSpeech lexinfo:noun ;
3  ...
```

Automatic grammar generation  Nouns
Alternative: Define additional classes

Winner ≡ ∃winner⁻¹. ⊤

Player ≡ Person ∨ ∃role. PlayerRole

<Declaration><Class IRI="#Player"/></Declaration>
<SubClassOf>
  <Class IRI="#Player"/>
  <Class IRI="#Person"/>
</SubClassOf>
<SubClassOf>
  <Class IRI="#Player"/>
  <ObjectSomeValuesFrom>
    <ObjectProperty IRI="#role"/>
    <Class IRI="#PlayerRole"/>
  </ObjectSomeValuesFrom>
</SubClassOf>
Relational nouns (2)

:capacity a lemon:LexicalEntry ;
  lexinfo:partOfSpeech lexinfo:noun ;
  lemon:canonicalForm [lemon:writtenRep «capacity»@en ;
    lexinfo:number lexinfo:singular ];
  lemon:synBehavior [ a lexinfo:NounPossessiveFrame ;
    lexinfo:copulativeArg :y ;
    lexinfo:possessiveAdjunct :x ] ;
  lemon:sense [ lemon:reference soccer:capacity ;
    lemon:subjOfProp :x ;

```
NP
  N capacity
  PP
dp_2 ↓ of

NP
  N capacity
  POSS
dp_2 ↓ ‘s

soccer:capacity(x, y)
(x, dp_2)
```

Automatic grammar generation     Nouns
lemon: A model for the lexicon-ontology interface

Automatic grammar generation
Verbs
Proper names
Nouns
Prepositions
Adjectives
Prepositions

:by a lemon:LexicalEntry;
lexinfo:partOfSpeech lexinfo:preposition;
lemon:canonicalForm [ lemon:writtenRep "by"@en ];
lemon:synBehavior [ lexinfo:complement :y ];
lemon:sense [ lemon:reference <soccer:byPlayer> ];
    lemon:subjOfProp :x;
    lemon:objOfProp :y ].

NP
  NP* PP
    P DP↓
      by

soccer:byPlayer(x, y)
(y, DP)
lemon: A model for the lexicon-ontology interface

Automatic grammar generation

Verbs
Proper names
Nouns
Prepositions
Adjectives
Adjectives

- **NP**
  - **ADJ**
  - **NP**

```
early
```

- **soccer:atMinute(x, m)**
- **m < 10**
Adjectives

\[ \text{soccer:atMinute}(x, m) \]
\[ \text{soccer:atMinute}(y, n) \]
\[ m < n \]
\[ (y, \text{DP}) \]
Adjectives

NP
  ├── ADJ
  │    └── NP
  └── earliest

NP
  ├── x
  │    └── m
  └── soccer:atMinute(x, m)

min(m)
Tomorrow
Useful References

- The Web site of the (Original) Lemon Model (http://lemon-model.net/)
- The lemon Cookbook (http://lemon-model.net/learn/cookbook.html)
- The Web site of LexInfo (http://lexinfo.net/index.html)

The W3C Ontology Lexicon Community Group (Ontolex) (https://www.w3.org/community/ontolex/) has developed a new Lexicon Model for Ontologies (https://www.w3.org/2016/05/ontolex/), which is influenced by the original Lemon Model we have discussed.